NVRAM

EEA index

FileStore
Description
Hardware
Error codes
NVRAM
Disc format
Accessing
Password file
E01 vs E01S
Rescuing

Intro
MDFS
Others
Clocks
Bridges
Interfaces
Misc h/w
Testing
Misc info

NVRAM

The FileStore contains a "CMOS RAM", which is a 146818-type fully programmable battery-backed memory device.
This device, hereafter referred to as NVRAM (non-volatile RAM), is used for three purposes:

1. To maintain a 'clock' during times when the server is switched off, so that upon power-up the server will know what the date and time are.
2. The storage of important system variables, such as the server's station number and the printer server 'name'.
3. Upon error, the error information is written to the NVRAM. I'm not entirely certain how you would go about extracting it, but it is there...

The NVRAM requires a continuous supply to keep the data intact. This low-current supply is provided by a NiCad battery that outputs 3.6V. The battery is charged by a constant current source from the +5V rail while the power is on.
When the power goes off, the battery powers the NVRAM and also pulls the `|CE` line high, which disengages the data and address lines in the device, offering a further reduction in the power required to maintain the configuration.

The small cylindrical crystal near the battery (in the **E01S**) provides the time base for the NVRAM. As is common for watches and the like, it operates at 32.768kHz. The variable capacitor near the crystal allows for fine-tuning and, when the server is undergoing testing, the test point (TP1, by the NVRAM chip), can be used to ensure that the ticker is running at *exactly* 1.024kHz.

The NVRAM's address register is mapped in to memory at `&FC00`, and the data register is mapped in to memory at `&FC04`.

NVRAM allocations

Byte(s)	Meaning
0 - 9	Real time clock (*internal function provided by the NVRAM chip*) **0** Seconds **1** (alarm seconds) **2** Minutes **3** (alarm minutes) **4** Hours **5** (alarm hours) **6** Day of the week (1-7; Sun=1) **7** Day of the month **8** Month **9** Year (range 0-99)

The RS data sheet (March 1997, component ref. 232-2605; stock no. 302-003) claims that the device offers automatic leap year compensation. I wonder what the in-built limits are, given that one of the data sheet examples refers to 1979, the Econet time ends in 2108, and the NVRAM has no century flag. How would it know if the year '0' is 1900, 2000, 2100, etc? Even more complicated, it appears from a cursory look at a disassembly of the MOS that the FileStore fudges the year value so 0 in the RTC means 1981!

10 Register A (*internal function provided by the NVRAM chip*)

bit 0 RS0
bit 1 RS1
bit 2 RS2
bit 3 RS3
These four Rate Selection bits select one of the 15 taps on the 22-stage divider to select the rate of the square wave on the SQW pin (pin 23, is connected to TP1) and/or the rate of the periodic interrupt. The FileStore sets this to provide an output on TP1 at 1.024kHz.

bit 4 DV0
bit 5 DV1
bit 6 DV2
These three DiVider chain bits select what crystal is connected to the NVRAM. The choice is 4.194304MHz, 1.048576MHz, or 32.768kHz. The FileStore sets this to 32.768kHz.

bit 7 UIP If set, the RTC cannot be accessed as Update In Progress.

11 Register B (*internal function provided by the NVRAM chip*)

bit 0 DSE Daylight Save Enable; may not be available on some 146818s.
bit 1 12/24 Select 12 hour or 24 hour mode.
bit 2 DM Is the Date Mode representing binary or BCD format?
bit 3 SQWE SQuare Wave Enable, output on pin 23.
bit 4 UIE Update-ended Interrupt Enable (cause IRQ after RTC updated?).
bit 5 AIE Alarm Interrupt Enable (cause IRQ from alarm match).
bit 6 PIE Periodic Interrupt Enable (cause IRQ from periodic timer).
bit 7 SET Inhibit RTC actions so RTC may be set.

The FileStore sets these flags to: no DST, 24 hour mode, binary date format, UIE enable, PIE and AIE both disabled, and clock to not be SET.

12 Register C (*internal function provided by the NVRAM chip*)

bit 0 not used
bit 1 not used
bit 2 not used
bit 3 not used
bit 4 UF Update-ended interrupt Flag.
bit 5 AF Alarm interrupt Flag.
bit 6 PF Periodic interrupt Flag.
bit 7 IRQF IRQ is ours Flag.

13 Register D (*internal function provided by the NVRAM chip*)

bit 0 not used
bit 1 not used
bit 2 not used
bit 3 not used
bit 4 not used

bit 5 not used
bit 6 not used
bit 7 VRT Valid Ram and Time

14 The server station number

15 255 - server station number (used as a check)

16 - 19 Reserved for use by the MOS

20 If error: Error code (refer to list)

21 If error: X register

22 If error: Y register

23 If error: Address MSB

24 If error: Address LSB
The effective address is thus `(?23 * 256) + ?24`.

25 - 27 If error: The 24 bit SIN

28 The maximum number of users that can log in concurrently.

29 The number of configured drives ***plus one***.

30 - 35 The printer server name (six characters maximum).
If ?30 = 0 then the printer server is disabled.

36 - 46 The username allowed to log in in maintenance mode.

47 Check byte for the above username - algorithm described below.

48 Printer page feeds - 0 is on, anything else means off.

49
Apparently if '0' (default) then the copyright check is on.

	The implication is that by setting this to non-zero, it is possible to disable the requirement for the string "(C)Acorn" to be present in a special area on the harddisc. I have not tried this, and I do not recall where I obtained this information, so if anybody can confirm whether or not this is correct...?
50 - 63	Unused. Software that is to run in the server that has a need of NVRAM space is requested to start allocation at byte 63 and work backwards.

Default values

If the NVRAM is detected to be invalid, then defaults will be set.

If the battery's charge is marginal, it is possible for bogus data to be stored in the NVRAM and the server will use sensible defaults but *not* clear the contents of the NVRAM. A suggested workaround is to use *!FSUtil* to "test" the NVRAM. The first operation it will do is read a copy of the current contents into memory. It will then write zeroes into NVRAM and attempt to read them back. ***As soon as !FSUtil starts to read back the data, you should switch your FileStore off***. Don't worry - *!FSUtil* will crash whinging about not being able to read back the information.
Now when you switch your FileStore on, it will detect that the NVRAM is blank, and will then write defaults, which are as follows:

Byte(s)	Meaning	Default
0 - 13	Time and date	**01:00:00 01-January-1900** *NetFS won't like this as it is a date before 1981!*
14	Server station	**254**
15	Server station check	**1**
16 - 19	Reserved	*Blank, zeroed.*
20 - 27	Error logout area	*Blank, zeroed.*
28	Maximum users	**80**
29	Drives (plus one)	**6** (all drives active)
30 - 35	Printer name	**PRINT** (padded with trailing spaces)

36 - 46	Maint. username	**SYST** (terminated with &0D and then nulls)
47	Username check byte	**&61** (algorythm described below)
48	Printer page feeds	**0** (do page feeds)
49	Copyright check	**0** (on)

The remaining bytes, 50 to 63, are not cleared by the FileStore as they are not used.

The FSUser check byte

The FSUser check byte is calculated by taking all ten bytes of the possible FSUser name (whether or not they are actually used in the name) and adding them.
If the value exceeds the range that would fit into a single byte, it is clipped to be within range (0 - 255) and one is added to the following byte.

In code terms, this would look like:

```
calcbyte% = 0
carry% = 0
FOR offset% = 36 TO 46
  calcbyte% = calcbyte% + cmosarray%?(offset%) + carry%
  IF calcbyte% >= &100 THEN
    calcbyte% = calcbyte% - &100
    carry% = 1
  ELSE
    carry% = 0
  ENDIF
NEXT
```

If you would like to test the algorithm yourself, then we know that for the following sequence:
`S Y S T <LF> <NUL> <NUL> <NUL> <NUL> <NUL>`
Which in hex is:
`53 59 53 54 0D 00 00 00 00 00 00`

The check byte value is **&61**. This is hardcoded into the FS ROM.

Here is a JavaScript-enabled way to calculate check bytes. Feel free to view the document source and look at the code.

FSUser name : SYST

Terminator : ○ Null (0x00) terminated
● Newline (0x0D) terminated

Calculate check byte

There is a caveat to be aware of. I do not currently know if the FSUser command clears existing data or if it simply writes new data.
For example, if you:

```
*FSUser JESSICA
*FSUser AMY
```

Would the FSUser name area look like *this*:

```
   A  M  Y  <-
   41 4D 59 0D 00 00 00 00 00 00
```

or like *this*:

```
   A  M  Y  <- I  C  A  <-
   41 4D 59 0D 49 43 41 0D 00 00
```

NVRAM read code

The code to perform the reading is:

```
  LDA  #5
  LDX  #(<address> MOD &100)
  LDY  #(<address> DIV &100)
  JSR  &FFF1 ; OSWORD
  RTS
```

Where *address* is where you want the result to be stored to (which should not be the same place you decide to poke your code into the server memory). As a BASIC function to prebuild this code segment, it would appear as:

```
  buffy%?0 = &A9
  buffy%?1 = 5
  buffy%?2 = &A2
  buffy%?3 = address% MOD 256
  buffy%?4 = &A0
  buffy%?5 = address% DIV 256
  buffy%?6 = &20
  buffy%?7 = &FFF1 MOD 256
  buffy%?8 = &FFF1 DIV 256
  buffy%?9 = &60
```

This code is poked into the FileStore at the address you have decided (with `*FSProt Off`!). You should then *Remote_JSR* to your code. Finally, peek the address that you have specified to store the result to. The result is at offset **+4**; so read back 6 bytes, the result is now in **buffy%?4**.

The code used by *!FSUtil* to read the NVRAM is:

```
SYS "Hourglass_On"
FOR loop% = 0 TO 63
  cmosbuffer%?loop% = FNcmosfetch(loop%)
NEXT
SYS "Hourglass_Off"
```

This reads all 64 bytes of NVRAM, calling the *cmosfetch* function, which is:

```
DEFFNcmosfetch(addr%)
  LOCAL rec%
```

```
  FOR rec%=0 TO 5 : buffer%?rec%=0 : NEXT
  ?buffer%=addr%
  PROCcmosassemble(5)
  SYS "Econet_DoImmediate", 2, &E800, fsstn%, fsnet%, cmoscode%, 10, 10, 2 TO erp%
  IF (erp% <> 0) THEN call error handler...
  SYS "Econet_DoImmediate", 2, &E900, fsstn%, fsnet%, buffer%, 6, 10, 2 TO erp%
  IF (erp% <> 0) THEN call error handler...
  SYS "Econet_DoImmediate", 3, &E800, fsstn%, fsnet%, 0, 0, 10, 2 TO erp%
  IF (erp% <> 0) THEN call error handler...
  SYS "Econet_DoImmediate", 1, &E900, fsstn%, fsnet%, buffer%, 6, 10, 2 TO erp%
  IF (erp% <> 0) THEN call error handler...
=buffer%?4
```

What happens here is the ten bytes of code is uploaded to &E800, and then six bytes are uploaded to &E900. This may seem unnecessary, but it is required in order to ensure that our data area is blanked. Once this has been performed, we execute the remote procedure call to JSR to address &E800. This will cause the code to be executed.
Finally, we read six bytes back from the data area.

The two 256 byte blocks of memory are &E800 and &E900, both of which are MOS code, but are part of the copy loop and, apparently, are not used after the boot sequence so it is theoretically safe to use these pages for user code.

The final thing is the *cmosassemble* function, which is:

```
DEFPROCcmosassemble(readwrite%)
  cmoscode%?0 = &A9
  cmoscode%?1 = readwrite%
  cmoscode%?2 = &A2
  cmoscode%?3 = &E900 MOD &100
  cmoscode%?4 = &A0
  cmoscode%?5 = &E900 DIV &100
  cmoscode%?6 = &20
  cmoscode%?7 = &FFF1 MOD &100
  cmoscode%?8 = &FFF1 DIV &100
  cmoscode%?9 = &60
ENDPROC
```

The value of *readwrite* should be **5** (as shown in these examples) to read the NVRAM, and **6** if you wish to write it.

All of this code is seriously unoptimised. It was designed to read or write bytes one by one from the NVRAM, and essentially a complete read or write operation means going through the motions 64 times!

An easy optimisation would be:

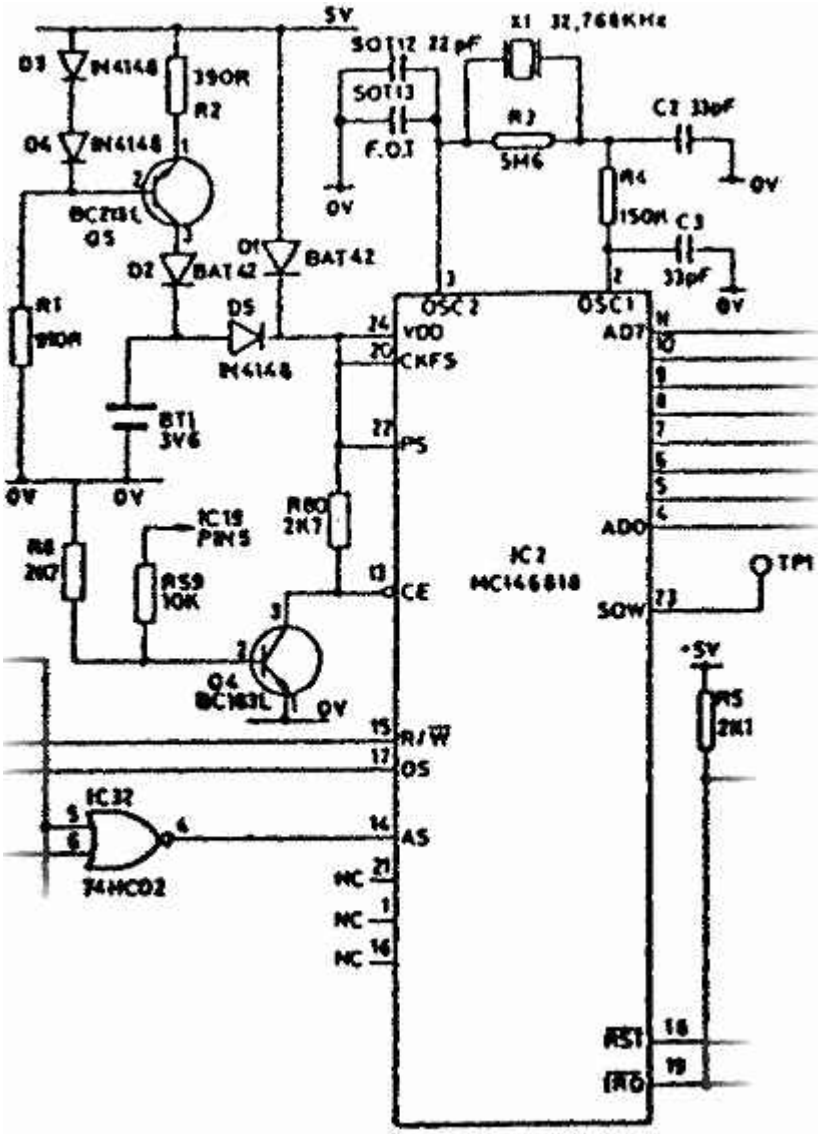
- Assemble code to read all 64 bytes at one time and write them one by one to the data area.
- Poke this code to the code area (&E800).
- Poke 64 bytes of null bytes to the data area (&E900).
- Execute the code on the FileStore.
- Read back 64 bytes from the data area.

This should complete in a *fraction* of the time it currently takes.

The 146818

The Real Time Clock / Non-Volatile RAM (RTC/NVRAM) chip used to hold the date and configuration in the FileStore is an interesting mixture of features and compactness. It uses a multiplexed bus interface where the address and data pins are one and the same, compatible with the 6800 mux circuit and also the 8085 bus. The first access sets the address and the second reads or writes data. This is handled by two pins, the Address Strobe (pin 14) and the Data Strobe (pin 17). Within the FileStore, the device's address and data are mapped into two different locations in memory (`&FC00` and `&FC04` respectively). The address decode logic will use this to determine whether to instruct the device to set an address or to access internal data.

This is better shown on the schematic below. This is from the **E01** as it is less cluttered than the **E01S'** circuit diagram. The address decode logic has been omitted as it would more than double the size of the picture.



Copyright © 2007 Rick Murray